(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 347 032 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication of patent specification: **13.12.95 Bulletin 95/50**

(51) Int. Cl.^{6}: **G06F 3/06**

(21) Application number: **89304146.7**

(22) Date of filing: **26.04.89**

(54) Record format emulation

(30) Priority: **20.05.88 US 197057**

(43) Date of publication of application: **20.12.89 Bulletin 89/51**

(45) Publication of the grant of the patent: **13.12.95 Bulletin 95/50**

(84) Designated Contracting States: **DE FR GB**

(56) References cited:
EP-A- 0 118 954
IBM TECHNICAL DISCLOSURE BULLETIN. vol. 26, no. 8, January 1984, NEW YORK US pages 4217 - 4232; R.L.HOFFMANN ET AL: 'NEW DIRECT-ACCESS STORAGE DEVICE FORMAT AND ATTACHMENT'

(73) Proprietor: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: **Menon, Moothedath Jaishankar**
6017 Montoro Place
San Jose California 95120 (US)

(74) Representative: **Burt, Roger James, Dr.**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

EP 0 347 032 B1

Description

This invention relates to record format emulation whereby data arranged in a first format can be recorded on a recording device that is formatted in a second format incompatible with the first format.

Information handling systems have used diverse formats for recording data on peripheral data recorders. When a given data recording format has been selected, the programming of the information handling system is designed to operate with the selected format. This limitation means that data recorders have to use the format contemplated in the programming, and vice versa. Some emulation of one recording format on another recording format has been achieved; such emulation has not enabled the programming to easily address the emulated format. That is, emulation software had to translate the addresses of the first format into different addresses of the second format. Such address translation not only obscures the record addresses of the first format when recorded over the second format but also can require substantial buffering and instruction execution resources.

Marilyn Bohl in INTRODUCTION TO IBM DIRECT ACCESS STORAGE DEVICES, publication SR20-4738, Science Research Associates, Inc. Chicago, Illinois, in Chapter 4 describes two data formats. The first is the well known count, key, data (CKD) format used on International Business Machines Co. (IBM) disk recorders, also termed disk files and direct access storage devices (DASD). Many large scale information handling systems employ CKD data format in their programming and recording. Bohl also describes fixed block (FB) recording, also termed fixed block architecture (FBA) format. In contrast, the CKD format enables efficient usage of disk space for large records by dividing disk track recording format in accordance with record size, whereas FBA format uses a set of identically-sized addressable areas (termed blocks), each of which can store up to some fixed amount of data, such as 512 bytes, 1024 bytes, 2048 bytes, etc. For small record sizes (such as less than 512 bytes) FBA may have some advantages over CKD. In any event, information handling systems are employed which have DASDs some of which use CKD while others use FBA formats. Many of the programs executed in the systems are designed to work with CKD format; therefore it is desired to enable CKD formatted data to be recorded on FBA devices. It is also desired to maintain the CKD record addressability for all data records stored in FBA formats.

Prior recording of CKD formatted data on FBA devices is described generally in the IBM publication GA33-1528-2, Third Edition, September, 1982, entitled "IBM 4321/4331 Processors Compatibility Features" in pages 1-1 through 1-5 and 2-1 through 2-3. In this format superposed recording (not true emulation), CKD formatted data as found on IBM 2311 and 2314 DASD volumes were recorded onto an IBM 3310 FBA recorder. This publication teaches that all gaps (non data containing areas interposed between various fields of the data format when recorded onto DASD) are removed for compacting the CKD formatted data onto the FBA recorder. This action obscures the CKD addressability of records in the CKD format. Entire tracks of CKD recorders (even though the track is not filled with data) are recorded onto FBA recorders. Mapping of the format-superposed recording is shown on page 2-3 of the publication. It is desired to provide a true emulation of CKD format upon FBA formatted recorders by maintaining CKD rotational addressability of CKD records stored in the FBA recorder.

To improve matters, the present invention as defined in claim 1 provides a method of writing, on a recording disk formatted in fixed-block architecture (FBA) in fixed-length blocks, count-key-data (CKD)-formatted binary data records that are separated by interrecord gaps, comprising the steps of: writing the CKD-formatted records into a buffer; using an emulator to reorient the records written into the buffer to create CKD-emulated records in FBA format by eliminating, from each CKD-formatted record before it is written to the disk, intrarecord gaps and other non-essential bytes; using the emulator to calculate the number and location of fixed-length blocks required to store each CKD-emulated record in a virtual track on the disk, where each virtual track consists of a predetermined number of fixed-length blocks at predetermined consecutive locations; and using the emulator to write to the disk the CKD-emulated records from the buffer in fixed-length blocks containing (i) header data indicating whether or not a CKD-emulated record begins in each such block, (ii) a number indicating byte displacement address of the record, and (iii) padding bytes between each set of adjacent CKD-emulated records including bytes corresponding to the interrecord gaps to ensure that the beginning of each respective CKD-emulated record has the same byte displacement from a reference location on a virtual track on the disk as each respective corresponding CKD-formatted record would have from an index if recorded on a physical track of a CKD-formatted disk.

Thus it is possible to reformat first formatted data records onto a second incompatible record format while maintaining the byte displacement relative position of the records of the first format in the second format and to provide addressing data of the first format when recorded in a second incompatible format by adding control information to the second format while maintaining the byte displacement address of the records to be the same in both formats wherein the addressing in the second format uses a virtual track concept.

As disclosed hereinafter, data in a first format which includes a control field (such as count and key fields in CKD formatted data) and a data field in each addressable record is modified to be stored as data in a second incompatible format while maintaining byte displacement addressability of the records of the first format when modified to be stored in the second format. Such modification creates an emulation of the first format into the second format by maintaining inter-record gaps in both formats while deleting intra-record gaps, error detection redundancies, physical parameter data, padding unique to the first format, during the modification. The data arrangement in the second format is independent of the first format such that records are split between addressable portions of the second format.

Control information is added to each of the addressable portions of the second format relating to one control field of one record stored in the addressable portion. The length of data in the first format is completely variable enabling a minimal number of the addressable portions to be consumed in storing the modified first formatted data in the second format.

One or more DASD tracks of CKD formatted data is emulated on an FBA format as a virtual track wherein blocks of the FBA format are the addressable portions of the second format. This aspect includes emulating virtual CKD disks consisting of many virtual tracks of data.

The present invention will be described further by way of example with reference to an embodiment thereof as illustrated in the accompanying drawings, in which:

Fig. 1 is a simplified diagram of an information handling system;

Fig. 2 is a diagram illustrating aspects of the emulation appropriate to the system illustrated in the Fig. 1;

Fig. 3 is a diagram showing of first and second data formats and the emulation of the first data format in the second data format; and

Fig. 4 is a simplified flow chart of machine operations employed.

Referring now more particularly to the drawings, like numbers and characters denote like parts and structural features shown in the various figures. An information handling system which advantageously employs this invention is shown as including a host processor 10 executing programs out of main memory 11. Usual procedures are employed for paging the programs from peripheral program storage (not shown) to main memory 11. Channel processor 12 (which is usually packaged in the same cabinet as host processor 10) executes channel programs stored in main memory 11 for executing desired input-output machine operations for host processor 10. Such channel programs are respective lists of channel control words (CCW's) as practiced in the so-called 370 architecture computers manufactured by IBM. Such usual channel program execution is represented by the channel execution portion 13 of channel processor 12. Channel processor 12 is connected via the usual cabling to the FBA adaptor 15 which in turn is connected to FBA device or DASD 16. Storing and retrieval of data in and from DASD 16 using FBA format follows the known peripheral data storage techniques. DASD 16 includes a plurality of co-rotating disks 17 each of which has one or two recording surfaces with a large plurality of addressable record tracks arranged in cylinders, as is well known.

Since many of the programs executed by host processor 10 employ the well known CKD data format, and such CKD formatted data cannot be directly recorded in DASD 16 – a CKD formatted DASD would normally be used. To make the programs of host processor 10 operate with the DASD 16 FBA format, emulator 20 is interposed between channel execution 13 and FBA adaptor 15 for emulating CKD formatted data in FBA format. Emulator 20 can be logic circuits in the form of semiconductor chips, programming either in ROM form, diskette form or paged into channel processor 12 from peripheral program storage. Such emulator may be lodged in main memory 11 for execution by the circuits of channel processor 12 or be in a separate program memory of channel processor 12.

CKD data is stored in FBA DASD 16 as a virtual track of data. The CKD data may be one or more CKD tracks of data. The number of bytes in the virtual track is not dependent on any physical track size in any CKD device or FBA device, rather it is the size of the CKD formatted data that determines the size of the virtual track. The number of bytes in the CKD data is known or can be easily calculated, see Bohl, supra. The following description assumes that a DASD disk 17 has been formatted to receive CKD formatted data. See the flow chart entitled "Steps to Initialize an FBA Disk for CKD Emulation", infra. A first step in creating the emulation is to calculate the number of blocks of DASD 16 track(s) that are needed to store the data. Let X be the number of bytes in the CKD data and BPB be the number of bytes storable in one addressable block of DASD 16. Further, it is desired to record a control information header HD in each of the blocks for facilitating direct access to any CKD record in the emulated or virtual track. Let CID be the number of bytes to be used for the header. Then B blocks of FBA format (B is (X/(BPB-CID)) rounded to the next higher integer). B blocks of FBA DASD 16 are allocated to store the CKD data. Such allocation follows usual procedures. It matters not whether one or more FBA tracks are employed, whether or not the allocation is fragmented with the fragments being link listed together (fragmentation reduces performance) and whether or not the allocation begins at an index or end of track (EOT) of DASD 16.

The modification of the CKD data is best generally understood by reference to Fig. 2 which shows the flow of the modification and Fig. 3 which shows the formatting of the emulation. For purposes of this description, emulator 20 is implemented as a set of programs 25 executable in channel processor 12. These channel programs are flow charted in the latter portion of this description. The invention is also easily practiced by incorporating programs constructed in accordance with the invention in a peripheral subsystem control unit, such as FBA adaptor 15. A virtual track buffer 26 is allocated as a part of main memory 11; in newer larger host processors, channel processor 12 has its own memory, in that instance buffer 26 is allocated as a part of the channel processor 12 memory. Each entry area 27 has a storage capacity equal to the storage capacity of FBA block. The left most two bytes of each entry store the later-described control information header HD and the remaining bytes of each buffer entry store the FBA formatted CKD data. HD is recorded in each FBA block in byte positions 0 and 1. CKD data is recorded in each FBA block beginning with byte position 2. Emulator 20 programs 25 process the CKD data to fit into the buffer 26 entries as shown in Fig. 3. The emulation is completed by emulator programs 25 doing a virtual rotational orientation of the virtual track using orientation table 28. This virtual orientation relates to the data stored in buffer 26. Once the CKD data is emulated on the FBA format within buffer 26, the buffer 26 contents flow through FBA adaptor 15 to DASD 16 for recording in the allocated FBA blocks. In a constructed embodiment, the contents of the allocated and emulator 20 formatted FBA blocks are read first from DASD 16 into buffer 26, whether or not such blocks contain any CKD data.

Fig. 3 shows the CKD and FBA formats with the CKD data being stored in true emulation in the FBA blocks. The CKD data format includes an end of track (EOT) indicator 30 (in a CKD DASD, EOT is index). In the virtual track EOT designates the beginning and end of the track, i.e. logically the virtual track is circular. Inter-record gap G1 at location 31 separates EOT from the first record HA 32 always found in a CKD track. A second inter-record gap G2' 33 separates HA 32 from the R0 record 34. R0 record 34 includes the control information field R0 count field (R0CF) 35 and the R0 data field R0DF 36. Third inter-record gap G3 37 separates record R0 34 from the first data record R1 40. Physical address mark A 41 is interposed between successive records within gap G3 in a CKD physical track. The CKD physical address information A is deleted from the emulation. CKD record 1 includes count field CF 42, key field KF 43 and data field DF 44. KF 43 is an optional field. The control information in record R1 40 is contained in CF 42 and KF 43. All subsequent records R2, et seq, in the CKD data use the illustrated well known format.

The FBA formatted track as it stores the CKD data for emulation is illustrated as FBA block 50 followed by FBA block 51. Block 50 is shown as storing the first portion of the CKD data. Block 50 can be at any circumferential location in the FBA track. FBID (fixed block physical identification) 52 is intermediate FBA blocks 50 and 51. FBID 52 stores the track address and the block address of the ensuing block. FBID 53 physically identifies block 50 location on the FBA disk. This FBA physical identification replaces the CKD physical identification used with the CKD format on DASD. Such substitution allows recovery processes which employ physical DASD locations. As will become apparent the CKD record address in byte displacement from the beginning of the data has a predetermined known relationship to the FBA physical address allowing directly and randomly addressing any of the CKD records as stored on the FBA formatted disk.

FBA block 50 illustrates the packing of the CKD formatted data for maintaining CKD byte displacement addressing of the CKD records. Note that the CKD byte displacement addressing on a CKD formatted disk is the rotational position of the record on the CKD track. True emulation requires that this same byte displacement occur in the virtual track as such track is realised physically in FBA. Block 50 has its first two bytes 0 and 1 occupied by the control header HD 55. Each HD includes a byte displacement pointer 56 which points to the first byte of the first occurring count field stored in the instant FBA block. In FBA block 50 pointer 56 points to the first byte of R0CF, the record R0 34 count field 35. If a given FBA block is storing no CKD count field, then the pointer is null and an indicator bit 59 is set to indicate no record begins (no count field) in the instant FBA block. Other control information beyond the present description may also be included in HD 55. Beginning at byte 2 (3 bytes displaced from the beginning of the FBA block), is gap G1 31. This inter-record gap has a correct number of gap indicating bytes (zeros, for example) that the first byte of record HA 32 begins at the same byte displacement within FBA block 50 as HA 32 is displaced from EOT 31 in the CKD format. CKD inter-record gap 33 is similarly replicated in FBA block 50 between HA 32 and R0 record 34 such that the first byte of R0CF 35 has the same displacement from byte 2 of FBA block 50 (beginning of the virtual CKD track) as it does from EOT in CKD format. All succeeding inter-record gaps 37 and 63 are similarly sized to achieve the byte displacement of the first byte of each CKD count field from the beginning of the CKD track and also from the beginning of the virtual track. Since the actual numerical computation is readily apparent from the two formats as described by Bohl, supra, the computation is omitted for purposes of brevity.

The physical relationship of the virtual track to the FBA blocks is relatively simple. Each FBA block is shown as capable of storing 2048 bytes. Since two bytes are used for HD 55, each FBA block stores 2046 bytes of the virtual track. FBA block 50 stores bytes 0 through 2045 while the next adjacent FBA block 52 stores bytes 2046 through 4091, the third FBA block (not shown) stores bytes 4092 through 6137, etc for all succeeding

FBA blocks (not shown) storing the CKD data as a virtual CKD track. The last FBA block in the allocated set of B (B is an integer) FBA blocks will probably not be completely used by the virtual track. In such instance, padding bytes fill the unused portion of the last allocated FBA block.

Much of the control information found in the CKD track is deleted and not recorded in the virtual CKD track of the physical FBA blocks. Each FBA block typically has its own error detection and correction (ECC) redundancy, its own padding bytes, its own physical parameter indications(such as defect information) and the like. The related DASD physical parameters found in the control portions of CKD recordings are deleted, all that are retained are those parameters necessary for logically defining the virtual track in sufficient precision to enable accessing records using the CKD byte address. The CKD count field is reduced from 28 bytes to 12 bytes. The retained 12 bytes contain CCHH (cylinder-head address), R (record number), KL(key length in bytes), and DL (data length in bytes). In addition, to identify the last record in the virtual track, emulator 20 adds one bit to the count field CF emulation. The one added bit LC 65, when set to 1, indicates that the count field is the last count field of the virtual CKD track. As records are added to the virtual track, the LC bit 65 that is set to unity is moved to the last added record. All other LC bits 65 are reset to zero. An LC bit 65 is also included in ROCF 35. Orientation table 28 indicates the virtual rotational CKD byte displacement information enabling CKD related programs to readily address and transfer data with buffer 26. Table 28 contains several entries as listed below:

EMULATOR ORIENTATION TABLE 28

OR	oriented bit (logical or virtual orientation)
CC	current cylinder address
CH	current head address
CS	current sector address
CFP	current field pointer
CFT	current field type (count, key or data)
NCP	next count field pointer
PCP	previous count field pointer
DL	CKD data field length in bytes
KL	key field length in bytes
NMR	no more records bit

Emulator 20 uses table 28 to logically emulate seeks and other DASD physical activity related to accessing CKD data on an actual CKD DASD. Fig. 4 is a simplified flow chart showing emulation activity related to data area accessing CKD data recorded and emulated on FBA DASD. A first operation in any DASD is to seek to the addressed track. Hence emulate seek step 70 (see textual flow chart "Execution of Seek Command") is performed in response to a SEEK channel command received from channel execution 13. Emulator 20 updates table 28 to reflect the CC and HH parameters passed with the SEEK command. Channel execution 13 generates the SEEK command from a SEEK CCW, as is known. A second step 73 is to obtain rotational orientation (OR = 0 above since there has been no rotational -- byte displacement identification -- orientation). A SET SECTOR channel command indicated by arrow 72 causes emulator 20 to emulate the SET SECTOR operations of a CKD device (see textual flow chart "Execution of Set Sector Command). This command merely identifies the rotational position (byte displacement) at which an ensuing data transfer command will be addressed. The OR bit of orientation table 28 is still zero. At this time no data has been moved from DASD 16 to buffer 26. When a SEARCH ID channel command is issued by channel execution at arrow 74, then host processor 10 is saying it wants data to be transferred. At this point in the sequence, emulator 20 accesses DASD 16 to stage data from DASD 16 to buffer 26 as indicated at step 75. Note that the ID portion of SEARCH ID gives identification in CKD address parameters; these parameters are not translated to independent addresses except for identifying which FBA block is storing the CKD count field CF related to the SEARCH ID command. This computation is simple, therefore no numerical examples are given.

As soon as FBA blocks are stored in buffer 26, emulator 20 accesses the buffer 26 in step 76 to emulate finding a CKD record on a CKD DASD. Once the desired record is found in buffer 26, a DE (device end) is sent to host processor 10 indicating orientation has occurred. The OR bit of orientation table 28 is set to unity.

Host processor 10 issues a data transfer command indicated by arrow 77 causing emulator 20 to emulate in step 78 a CKD read at from DASD by transferring data between buffer 26 and main memory 11. During such a data transfer, emulator 20 continues to stage FBA blocks from DASD 16 into buffer 26 simulating disk rotation in the buffer. Usually, a host processor reads or writes a large plurality of records beginning with a first addressed record. This activity continues until the end of the virtual track (end of the CKD data) irrespective of the index marks related to FBA operations with DASD 16, a received channel command indicates that the data transfer is stopping or an end of the channel operation is indicated. This continued staging independent of the host processor accesses is indicated by arrow 79.

Below are textual flow charts showing the details of the above described operations. The Glossary following the flow charts indicate the meaning of abbreviations used. Any programming language or logic hardware may be used to implement these flow charts. In a practical embodiment, other channel command execution commands in addition to what are illustrated may be used to complete a machine.

FLOW CHARTS SHOWING THE LOGIC OF EMULATOR 20

The textual flow charts below show the machine steps in channel processor 12 which illustrate this embodiment of the invention. These steps do not describe a complete machine design as an understanding of such other programs and structures found in a complete machine are not necessary to practice the present invention.

FLOW CHARTS OF CHANNEL COMMAND EXECUTION

including called subroutines

EXECUTION OF SEEK COMMAND

```
Validate received command
Verify that filemask allows seeks
Read seek parameters from received command, verify format
Validate cylinder (CC) and head (HH) addresses
IF seek needed (current CCHH does not equal requested CCHH), then continue, else go to step labelled "CEDE
to channel"
Save CCHH and apply same to seek circuits
Set "need to seek" bit 100 NS = 1
Set "oriented bit" to OR = 0
CEDE to channel; check chaining and proceed
```

EXECUTION OF SET FILE MASK COMMAND

```
Validate received command
Read filemask parameters from received command
Save filemask
Set "oriented bit" to OR = 0
CEDE to channel; check chaining and proceed
```

EXECUTION OF SET SECTOR COMMAND

```
Validate received command
Read parameters from received command
Validate sector number of command and save it as S
Set "oriented" bit OR = 0
CEDE to channel; check for chaining and continue.
```

EXECUTION OF SEARCH ID COMMAND

```
Validate received command
IF OR = 0, call GET ORIENTED subroutine
Read parameters from received command
```

DO ONE OF FIVE CASES

```
IF CFT = CNT, then do CASE I, else
IF CFT = R0CNT, then do CASE II, else
IF ID points to a non-count field of a record other than last record OR if ID points to HA and R0 exists, do CASE
```

```
III, else
IF (oriented beyond last record OR to non-count field of last record) AND not a multi-track operation, do CASE IV, else
IF (oriented beyond last record OR to non-count field of last record) AND multi-track operation is being performed, do CASE V

END OF FIVE CASES

Set SIZE = number of bytes of received parameters
Compare SIZE, starting at PTR with parameters
Send results to channel, successful or unsuccessful compare;
        if SIZE < 5, on successful compare, save as SIZE 5.
Do ending status

STEPS OF CASE I

PTR = CFP
CFP = CFP + 12
IF KL = 0; then CFT = DF, else CFT = KF

STEPS OF CASE II

PTR = CFP
CFP = CFP + 12
IF KL = 0, then CFT = R0DF, else CFT = R0KF

STEPS OF CASE III

PTR = NCP
KL = key length of record at pointer PTR
DL = data length of record at pointer PTR
IF CFT = HA field, then PCP = CFP, else PCP = next count (calculate next count pointer)
CFP = PTR + 12
IF KL = 0, then CFT = DF; else CFT = KF
NCP = calculate next count pointer(call subroutine)
IF LC = 1, then NMR = true (1), ELSE NMR = false (0)

STEPS OF CASE IV

IF EOT (index passed) = 1, then ERROR; else EOT = 1 (index passed
BB = (T*C + H)*B
BBLAST = (T*C + H)*B + (BPS*S/BBB) - 1 (see Glossary for BBB) Set S to be £ G1/BPS and < (G1+G2'+SIZE(HA)+PADH + ECCH))/BPS
Call GET ORIENTED at step 4
IF GET ORIENTED = 0, then indicate no record found, else
record found
PTR = CFP
CFP = CFP + 12
if KL = 0, then CFT = R0DF, else CFT = R0KF

STEPS OF CASE V

Set H = H + 1, IF new H £ T, signal error
Set S £ (G1/BPS) but < (G1 + G2' + SIZE(HA) + PADH +
ECCH)/BPS
Call GET ORIENTED beginning at step 2
IF no record found, return to Set S, else continue
PTR = CFP
```

```
CFP = CFP + 12
IF KL = 0, then CFT = R0DF, else CFT = R0KF

EXECUTION OF READ DATA COMMAND

Validate received command
IF previous CMD was SEARCH that compared equal; PO = 1
IF OR = 0, call GET ORIENTED
IF oriented to non-R0 record OR (oriented to R0 and PO =1, then do CASE A, else
IF oriented to R0 AND (previous CMD was not SEARCH ID nor SEARCH KEY and R1 is present, do CASE B,
else
IF oriented to HA AND R0 and R1 are present, then do CASE C,
IF EOT = 1 and not a multi-track operation, then do CASE D, else
IF EOT = 1 and it is a multi-track operation, then do CASE E

END CASES

Read number of bytes L from location PTR
IF L=0, UC to channel with CEDE, else CEDE to channel

STEPS OF CASE A

PTR = CFP
IF CFT = KF OR R0KF, then PTR = PTR + KL, else
IF CFT = CNT or R0CF, then PTR = PTR + 12 + KL
DO STEPS Y

STEPS Y

L = DL
CFP = NCP
PCP = calculate next CNT from PCP
IF NMR = 1, then CFT = index (EOT), else DO
    DO LOOP
    CFT = CNT
    KL = KL of CFP
    DL = DL of CFP
    NCP = calculate from CNT of CFP
    IF LC = 1, then NMR = 1, else NMR = 0

STEPS OF CASE B

CFP = NCP
L = DL of CFP
PTR = CFP + 12 +KL
PCP = CFP
CFP = calculate next CNT from CNT of CFP
DO STEPS X

STEPS X

IF LC = 1 for PCP, then CFT = EOT, else DO
    DO LOOP
    CFT = CNT
    KL = KL of CFP
    DL = DL of CFP
    NCP = calculate next CNT from CFP
    IF LC = 1, then NMR = 1, else NMR = 0
```

```
          END DO

STEPS OF CASE C

PTR = calculate next CNT from NCP
L = DL of PTR
CFP = PTR
PTR = PTR + 12 + KL
PCP = CFP
DO STEPS X

STEPS OF CASE D

IF EOT = 1, set ERROR, else set EOT = 1
BB = (T*C + H)* B
BBLAST = (T*C +H)* B + (BPS*S/BBB) - 1 (BBB - see Glossary)
Set S £ (G1 + G2' + SIZE(HA) + PADH + ECCH)/BPS
Set S < (G1 +G2' + SIZE(HA) + PADH + ECCH + G2 + G3 + SIZE(CF) + PADC + ECCC + 8 + ECCD)/BPS
Call GET ORIENTED, start at step 4
IF OR = 0, no record found, else record found
PTR = CFP + KL + 12
DO STEPS Y

STEPS OF CASE E

Set H = H + 1, IF new H £ T, set ERROR
Set S £ (G1 + G2' + SIZE(HA) + PADH + ECCH)/BPS
Set S < (G1 + G2' + SIZE(HA) + PADH + ECCH + G2 + G3 +SIZE(CF) + PADC + ECCC + 8 + PADD +
ECCD)/BPS
Call GET ORIENTED, start from step 2
IF NRF = 1, (no record is found), return to first step of CASE E, else continue
PTR = CFP + KL + 12
DO STEPS Y

END OF CASES A THROUGH E

Send L bytes to channel, starting from PTR
IF L = 0, CEDE UC to channel, else CEDE to channel

EXECUTION OF WRITE DATA COMMAND

Validate received command
CKD data is received from host processor and stored in a
separate buffer area of main memory 11, the following steps operate on the stored CKD data after it is converted
as shown in Fig. 3.
PTR = CFP
IF CFT = KF or R0KF, then PTR = PTR + KL
DO STEPS Y
Store L bytes of CKD data from host in buffer 26 at PTR.
Write updated or new FBA blocks (such as 50 and 51) into DASD 16.
Upon writing all FBA blocks to DASD 16, send CEDE to the channel.
Proceed to other operations.

GET ORIENTED SUBROUTINE

This subroutine manages orientation table 28.
IF no SEEK command this chain of commands, CC = CH = 0
IF no SET Sector command this chain of commands CS = 0
```

STEP 2

```
Identify the first FBA block to be staged to buffer 26, i.e. which stores the first record to be transferred. first FBA block address = (T*C + H)*B + (BPS*CS/BBB) rounded to next lower integer
Identify last FBA block to be staged to buffer 26 from DASD 16.
last FBA block address = (T*C + H)*B + (B-1)
```

STEP 4

```
Read FBA blocks identified above and all intermediate FBA blocks into buffer 26 as shown in Fig. 2. Keep HD's separate from data in FBA blocks (bytes 2-2046)
Examine HD of first FBA block;
    IF no pointer to a CF in the FBA block, examine HD's succeeding FBA blocks until the first count field is located.
    IF EOT is reached without a count field, then stage FBA blocks from beginning of the virtual track to the first FBA block to find a count field, if no count field is found, signal UC to channel.
Set FBA block NR (block having first count field) = first FBA block - (T*C + H)*B
Set Z = (NR*BBB) + offset pointer from HD (number of bytes the first byte of the first count field is inside the FBA block NR
IF BPS*S <= Z, then orientation is to HA or a count field beyond sector S, skip to "step 9), else continue
IF LC = 0, then:
    Set Z' = Z + 12 + KL + DL + PADC + ECCC + PADD + ECCD + G2 + G3
IF KL is not zero, then Z = Z' + PADK + ECCK + G2, else Z = Z'
(Above assumes search is between first record and EOT)
```

STEP 9

```
Set NS =0 (no need to seek)
Update orientation table 28
    CFP = Z
    KL = KL in count field of record
    DL = DL in count field of record
    CC = cylinder address in count field of record
    CH = head address in count field of record
    CFT = IF S*BPS < G1, to HA; else IF G1 < S*BPS < G1 + G2' + SIZE(HA) + PADH + ECCH, to R0 count field R0CF, else CF of data record "n".
    OR = 1
    IF LC = 1, then set NMR = 1, else NMR = 0
Return to caller
```

CALCULATE NEXT COUNT FIELD POINTER SUBROUTINE

```
Pointer PTR2 points to next count field location in the emulated virtual track.
    IF PTR = G1, then PTR2 = PTR + 12 + PADH + ECCH + G2' else PTR2 = PTR + 12 + KL(record pointed to by PTR) + DL(record pointed to by PTR) + PADC + ECCC + PADD + ECCD + G2 + G3
    IF KL(of record pointed to by PTR) ≠0;
        then PTR2 = PTR2 + G2 + PADK + ECCK
Return PTR2 to caller
```

PROGRAMS FOR INITIALISING FBA DISK FOR EMULATION

NUMBER OF FBA BLOCKS FOR CKD DISK

Let X = number of bytes in a CKD track
Let B = X/BBB (number of blocks used for each CKD track Number of tracks per CKD device is product of number of cylinders times the number of tracks per cylinder. Number of blocks for each CKD device is the product of B times the number of tracks in the CKD device.

STEPS TO INITIALISE AN FBA DISK FOR CKD EMULATION

Let CYL = the number of cylinders of emulated CKD device

First initialise all FBA blocks on FBA disk to be used for emulation to store the hexidecimal pattern 0008 in bytes 0 and 1 of each FBA block to be used. In addition, every Bth block contains a logical HA and R0 record. This inclusion is described below.

DO for all of the FBA blocks to be initialised with HA and R0:

Set buffer 26 to one FBA block which will be repeatedly used.

In bytes 0 and 1 store G1 in first 12 bits and all 0's in last 4 bits.

At byte offset of G1 + 2 in the respective blocks, write HA with zero's in flag byte, CC = cylinder number incremented from 0 to CYL - 1; HH = track number incremented from zero to T - 1, R byte (record number) = zero; KL = 0; DL = 0, LC = 0 and pad bytes = 0's

At byte offset in the respective blocks of G1 + G1' + 2(HA) + ECCH + PADH create R0 as above for HA except that DL = 8 and LC is all ones.

Write block from current buffer to FBA disk at appropriate blocks separated by B - 1 blocks on the FBA disk.

STEPS FOR WRITING CKD TRACK INTO B FBA BLOCKS

This flow chart relates to copying data from a CKD track to the B FBA blocks on the FBA disk.

Make buffer 26 with B buffer entries (number 0 through B-1) Set B flag bits BF 101 in main memory initialised to 0 (false); when for a given FBA block (buffer 26 entry), its flag BF = 1, then control information has been recorded for the FBA block related to the flag bit. In buffer 26 store the binary pattern 00001000 in the second byte of all respective HD's. This pattern indicates no data has been recorded in the represented FBA block.

CB 102 points to the next byte to be written into buffer 26 entry. Minimum value is 2 (jump over HD bytes 0 and 1).

The CKD track has been copied as a CKD track in main memory 11 having fields HA, R0CF, R0DF, CF1, KF1, DF1, etc.

Memory pointers are all zeros.

The count fields have only 12 bytes rather than 28 -- CKD physical parameters etc have been deleted.

Each time Write is called, bytes are written starting from CB, the HD bytes are always skipped. If a count field is being written and its corresponding flag BF is off, its flag BF is set on. The corresponding HD is made to contain an offset to the written count field. CB is always updated.

STEPS of writing CKD into a buffer entry.

```
CB = 2
Write G1 zero bytes (update CB each byte recorded)
Write HA
Write G2' + Size(HA) - 12 + PADH + ECCH of zero bytes
Write R0CF
Write R0DF
Write G2 + G3 + Size(CNT) - 12 + PADC + ECCC + PADD + ECCD of zero bytes.
    Repeat for each CKD record other than record R0:
Write CF
Write KF
Write DF
Write G1 + G2 + G3 + Size(CNT) - 12 + PADC + ECCC + PADK + ECCK + PADD + ECCD of zero bytes.
    Write buffer 26 to the B FBA blocks on DASD 16
```

GLOSSARY OF SELECTED TERMS

B	Number of FBA blocks involved in one CKD virtual track
BB	Block number of first FBA block to be staged into the current buffer. BBB The value BPB - CID but not BB
BBLAST	The FBA block number of the last block to be staged into the current buffer in on CKD staging operation.
BF	Flag related to block b indicating whether or not formatted
BPB	Number of bytes per FBA block

BPS	Number of bytes per sector in CKD data format
CB	Pointer to current byte being processed
CC	Cylinder number or address of CKD device being emulated
CCW	Channel command (control) word
CE	Channel End signal, indicates channel can be freed
CEDE	Channel End, Device End, indicates to channel operations completed
CF	Count field in CKD format record
CFP	Current field pointer
CFT	Current field type
CID	The number of bytes for the header in an FBA device
CKD	Count, key, data
CMD	Channel command
CYL	Number of cylinders in one CKD DASD
DE	Device end, signal to channel that device has completed operation
DF	Data field of CKD format record
DL	Length of data field DF, a field in count field CF
ECC	Error detection and correction redundancy
ECCC	Number of ECC bytes in count field CF
ECCD	Number of ECC bytes in data field DF
ECCH	Number of ECC bytes in HA
ECCK	Number of ECC bytes in key field KF
EOT	End of track, relates to index of physical DASD
FBA	Fixed Block Architecture
Gx	Gap length in bytes where x = 1 through 3, CKD format
HH	Head or surface address of DASD (CKD or FBA)
HA	Home Address record of CKD format
HD	Header (control field) in FBA block for CKD emulation
KF	Key field of CKD format
KL	Key length, indicator in count field CF showing KL
NCP	Next count field pointer
NR	Next CKD record to be accessed
NRF	No record found meeting search criteria
NS	Need to seek bit or flag
PADC	Number of pad bytes in count field CF
PADD	Number of pad bytes in data field DF
PADH	Number of pad bytes in HA
PADK	Number of pad bytes in key field KF
PCP	Previous count field pointer
PDS	Partitioned data
PO	Previously oriented logically
PTR	Pointer used to point to field/item being processed
R	Record number, field is in count field CF
S	Sector number
SIZE	Size of field in bytes on CKD device being emulated (field identified following term)
T	Number of tracks in one cylinder of DASD
UC	Unit Check, a signal to the channel indicating an error
VTOC	Volume table of contents in DASD

Claims

1. A method of writing, on a recording disk formatted in fixed-block architecture (FBA) in fixed-length blocks, count-key-data (CKD)-formatted binary data records that are separated by interrecord gaps, comprising the steps of:

    writing the CKD-formatted records into a buffer;

    using an emulator to reorient the records written into the buffer to create CKD-emulated records in FBA format by eliminating, from each CKD-formatted record before it is written to the disk, intrarecord gaps and other non-essential bytes;

using the emulator to calculate the number and location of fixed-length blocks required to store each CKD-emulated record in a virtual track on the disk, where each virtual track consists of a predetermined number of fixed-length blocks at predetermined consecutive locations; and

using the emulator to write to the disk the CKD-emulated records from the buffer in fixed-length blocks containing (i) header data indicating whether or not a CKD-emulated record begins in each such block, (ii) a number indicating byte displacement address of the record, and (iii) padding bytes between each set of adjacent CKD-emulated records including bytes corresponding to the interrecord gaps to ensure that the beginning of each respective CKD-emulated record has the same byte displacement from a reference location on a virtual track on the disk as each respective corresponding CKD-formatted record would have from an index if recorded on a physical track of a CKD-formatted disk.

2. The method as claimed in claim 1, including the step of:
providing, between each of the blocks, identification indicia containing the physical track address and location on the disk of the block which follows said indicia.

3. The method as claimed in claim 1 or claim 2, including the step of:
providing a binary indicator identifying the end of the last emulated record on a virtual track.

4. The method as claimed in claim 3, including the step of:
moving the binary indicator progressively as additional emulated records are added for identifying a then current last emulated record.

5. The method as claimed in any preceding claim, including the step of:
transmitting the CKD data from a programmed processor without buffering to the emulator without the said non essential bytes, which non essential bytes would have been generated if the CKD data had been sent to a CKD controller for recording in a physical track on a CKD-formatted disk.

6. An apparatus for recording count-key-data (CKD)-formatted binary data records that are separated by interrecord gaps, comprising:
a recording disk (17) formatted in fixed-block architecture (FBA) with fixed-length data blocks;
a buffer (26) for receiving the CKD-formatted records;
an emulator (25) connected to the buffer for generating from each received CKD-formatted record, and storing in the buffer, a CKD-emulated record in FBA format;
controller means (15) operatively connected to the emulator and the disk for transmitting CKD-emulated records from the buffer under the control of the emulator to the disk for recording thereon in virtual tracks with fixed-length blocks at predetermined consecutive locations; and
said emulator being operative to (i) remove from each CKD-emulated record bytes not essential for recreating in CKD format the data of that respective CKD-formatted record, (ii) calculate the number and location of fixed-length blocks required to store each CKD-emulated record in a virtual track on the disk, (iii) provide, in each block, a number relating to byte displacement address and a binary indicator indicating whether a CKD-emulated record begins in such block, and (iv) provide, between adjacent CKD-emulated records, padding bytes in number sufficient to assure that the beginning of each respective CKD-emulated record has the same byte displacement from a single reference location on a virtual track on the FBA-formatted disk as each respective corresponding CKD-formatted record would have from an index if recorded on a physical track of a CKD-formatted disk.

Patentansprüche

1. Ein Verfahren, um in eine Aufzeichnungsplatte, die in einer Festblockarchitektur (FBA) in Blöcken mit fester Länge formatiert ist, zähl-, schlüssel-, datenformatierte (CKD) binäre Datensätze zu schreiben, die durch Satzzwischenräume getrennt sind, wobei das Verfahren Schritte enthält, um
die CKD formatierten Sätze in einen Puffer zu schreiben;
einen Emulator zu verwenden, um die in den Puffer geschriebenen Sätze neu anzuordnen, um CKD emulierte Sätze in der FBA Struktur zu erstellen, indem aus jedem CKD formatierten Satz Satzzwischenräume und andere unwesentliche Bytes gelöscht werden;
den Emulator zu benutzen, um Anzahl und Speicherstelle der Blöcke mit fester Länge zu berechnen, die erforderlich sind, um jeden CKD emulierten Satz in einer virtuellen Spur auf Platte zu speichern, wobei

jede virtuelle Spur aus einer zuvor bestimmten Anzahl Blöcke mit fester Länge in zuvor bestimmten aufeinanderfolgenden Speicherstellen besteht;
und den Emulator zu benutzen, um auf die Platte die CKD emulierten Sätze aus dem Speicher in Blöcke mit fester Länge zu schreiben, die (i) Anfangsdaten enthalten, die angeben, ob ein CKD emulierter Satz in jedem Block beginnt oder nicht, (ii) eine Nummer enthalten, welche die Byte-Distanzadresse des Satzes angibt und (iii) Auffüllbytes zwischen jedem Satz von benachbarten CKD emulierten Sätzen enthalten, die Bytes enthalten, welche den Satzzwischenräumen entsprechen, um sicherzustellen, daß der Anfang von jedem entsprechenden CKD emulierten Satz die gleiche Byte-Distanz von einer Referenzspeicherstelle in einer virtuellen Spur auf der Platte wie jeder entsprechende CKD formatierte Satz in einem Index hätte, wenn dieser auf einer physischen Spur einer CKD formatierten Platte aufgezeichnet worden wäre.

2. Das Verfahren wie in Anspruch 1 angemeldet, das den Schritt enthält, um
zwischen jedem der Blöcke Identifikationsindizes bereitzustellen, welche die physische Spuradresse und Speicherstelle auf der Platte des Blockes enthalten, der diesen Indizes folgt.

3. Das Verfahren wie in Anspruch 1 oder Anspruch 2 angemeldet, das den Schritt enthält, um
einen binären Indikator bereitzustellen, der das Ende des zuletzt emulierten Satzes in einer virtuellen Spur identifiziert.

4. Das Verfahren wie in Anspruch 3 angemeldet, das den Schritt enthält, um
den binären Indikator progressiv zu bewegen, da zusätzliche emulierte Sätze hinzugefügt werden, um dann den aktuell zuletzt emulierten Satz zu identifizieren.

5. Das Verfahren wie in irgendeinem vorhergehenden Anspruch angemeldet, das den Schritt enthält, um die CKD Daten aus einem programmierten Prozessor ohne Pufferung im Emulator, ohne die unwesentlichen Bytes zu übertragen, wobei die unwesentlichen Bytes generiert werden, wenn die CKD Daten an einen CKD Controller gesendet wurden, um in einer physischen Spur auf einer CKD-formatierten Platte aufgezeichnet zu werden.

6. Eine Vorrichtung, um zähl-, schlüssel-, datenformatierte (CKD) binäre Datensätze, die durch Satzzwischenräume getrennt sind, aufzuzeichnen, wobei die Vorrichtung enthält:
eine Aufzeichnungsplatte (17), die in einer Festblockarchitektur (FBA) in Datenblöcken mit fester Länge formatiert ist;
einen Puffer (26), um die CKD-formatierten Sätze zu empfangen;
einen Emulator (25), der mit dem Puffer verbunden ist, um von jedem empfangenen, CKD-formatierten Satz einen CKD-emulierten Satz in FBA Struktur zu generieren und diesen im Puffer zu speichern;
Controllermittel (15), die funktionsmäßig mit dem Emulator und der Platte verbunden sind, um die CKD-emulierten Sätze aus dem Puffer mittels der Steuerung des Emulators auf die Platte zu übertragen, um diese auf virtuellen Spuren in Blöcken mit fester Länge in zuvor bestimmten aufeinanderfolgenden Speicherstellen aufzuzeichnen; und
der Emulator in der Lage ist, (i) aus jedem CKD-emulierten Satz unwesentliche Bytes zu entfernen, um die Daten aus diesem entsprechenden CKD-formatierten Satz in der CKD Struktur neu zu erstellen; (ii) die Anzahl und Speicherstelle von Blöcken mit fester Länge zu berechnen, die benötigt werden, um jeden CKD-emulierten Satz in einer virtuellen Spur auf der Platte zu speichern; (iii) in jedem Block eine Nummer bereitzustellen, die zu der Byte-Distanzadresse gehört und einen binären Indikator, der angibt, ob ein CKD-emulierter Satz in einem solchem Block beginnt; und (iv) zwischen angrenzenden CKD-emulierten Sätzen Auffüllbytes in ausreichender Anzahl bereitstellt, um sicherzustellen, daß der Anfang von dem jeweiligen CKD-emulierten Satz die gleiche Byte-Distanz von einer einzelnen Referenz-Speicherstelle in einer virtuellen Spur auf der FBA-formatierten Platte wie der jeweilige entsprechende CKD-formatierte Satz aus einem Index hat, wenn dieser auf einer physischen Spur einer CKD-formatierten Platte aufgezeichnet wird.

Revendications

1. Procédé d'écriture, sur un disque d'enregistrement formaté dans une architecture à bloc fixe (FBA) dans des blocs de longueur fixe, des enregistrements de données binaires formatées par données à code de compte (CKD) qui sont séparés par des intervalles entre enregistrements, comprenant les étapes sui-

vantes:

écriture des enregistrements formatés CKD dans un tampon;

utilisation d'un émulateur pour réorienter les enregistrements écrits dans le tampon pour créer des enregistrements émulés CKD en format FBA en supprimant, de chaque enregistrement formaté CKD avant qu'il soit écrit sur le disque, des intervalles entre enregistrements et autres multiplets non essentiels;

utilisation de l'émulateur pour calculer le nombre et l'emplacement de blocs de longueur fixe nécessaires pour stocker chaque enregistrement émulé CKD dans une piste virtuelle sur le disque, où chaque piste virtuelle consiste en un nombre prédéterminé de blocs de longueur fixe en des emplacements consécutifs prédéterminés; et

utilisation de l'émulateur pour écrire sur le disque les enregistrements émulés CKD à partir du tampon dans des blocs de longueur fixe contenant (i) des données d'en-tête indiquant si oui ou non un enregistrement émulé CKD commence dans chacun de ces blocs, (ii) un nombre indiquant l'adresse de déplacement de multiplet de l'enregistrement, et (iii) des multiplets de bourrage entre chaque série d'enregistrements émulés CKD adjacents comportant des multiplets correspondant aux intervalles entre enregistrements pour assurer que le début de chaque enregistrement émulé CKD respectif a le même déplacement de multiplet par rapport à un emplacement de référence sur une piste virtuelle sur le disque que chaque enregistrement formaté CKD respectif correspondant aurait à partir d'un index s'il était enregistré sur une piste physique d'un disque formaté CKD.

2. Procédé selon la revendication 1, comportant l'étape suivante:

fourniture, entre chacun des blocs, d'indices d'identification contenant l'adresse et l'emplacement de la piste physique sur le disque du bloc qui suit lesdits indices.

3. Procédé selon la revendication 1 ou la revendication 2, comportant l'étape suivante:

fourniture d'un indicateur binaire identifiant la fin de l'enregistrement émulé en dernier sur une piste virtuelle.

4. Procédé selon la revendication 3, comportant l'étape suivante:

déplacement de l'indicateur binaire progressivement lorsque des enregistrements émulés supplémentaires sont ajoutés pour l'identification de l'enregistrement émulé en dernier alors en cours.

5. Procédé selon l'une quelconque des revendications précédentes, comportant l'étape suivante:

transmission des données CKD à partir d'un dispositif de traitement programmé sans tamponnage de l'émulateur sans lesdits multiplets non essentiels, ces multiplets non essentiels ayant été générés si les données CKD avaient été envoyées à un dispositif de commande CKD pour l'enregistrement dans une piste physique sur un disque formaté CKD.

6. Appareil pour l'enregistrement d'enregistrements de données binaires formatés (CKD) à données de codage de compte qui sont séparés par des intervalles entre enregistrements, comprenant:

un disque d'enregistrement (17) formaté en architecture à bloc fixe (FBA) avec des blocs de données à longueur fixe;

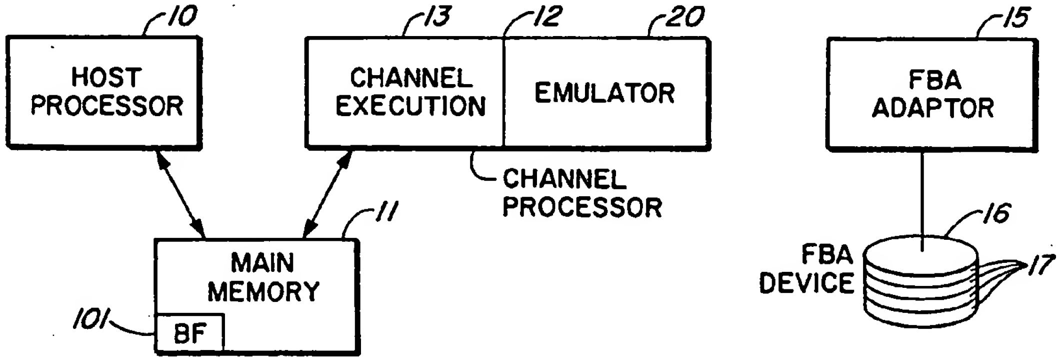
un tampon (26) pour recevoir les enregistrements formatés CKD;

un émulateur (25) connecté au tampon pour générer à partir de chaque enregistrement formaté CKD reçu, et stocker dans le tampon, un enregistrement émulé CKD en format FBA;

un moyen de commande (15) connecté de manière opérationnelle à l'émulateur et au disque pour transmettre des enregistrements émulés CKD à partir du tampon sous la commande de l'émulateur au disque pour l'enregistrement sur celui-ci dans des pistes virtuelles, avec des blocs de longueur fixe en des emplacements consécutifs prédéterminés; et

ledit émulateur étant opérationnel pour (i) supprimer de chacun des enregistrements émulés CKD les multiplets non essentiels pour recréer en format CKD les données de cet enregistrement respectif formaté CKD, (ii) calculer le nombre et l'emplacement de blocs de longueur fixe nécessaires pour stocker chaque enregistrement émulé CKD dans une piste virtuelle sur le disque, (iii) fournir, dans chaque bloc, un nombre relatif à l'adresse de déplacement de multiplet et un indicateur binaire indiquant si un enregistrement émulé CKD commence dans ce bloc, et (iv) fournir, entre des enregistrements adjacents émulés CKD, des multiplets de bourrage en nombre suffisant pour assurer que le début de chaque enregistrement respectif émulé CKD a le même déplacement de multiplet par rapport à un seul emplacement de référence sur une piste virtuelle sur le disque formaté FBA que chaque enregistrement formaté CKD correspondant et respectif aurait par rapport à un index s'il était enregistré sur une piste physique d'un disque formaté CKD.

FIG. 1



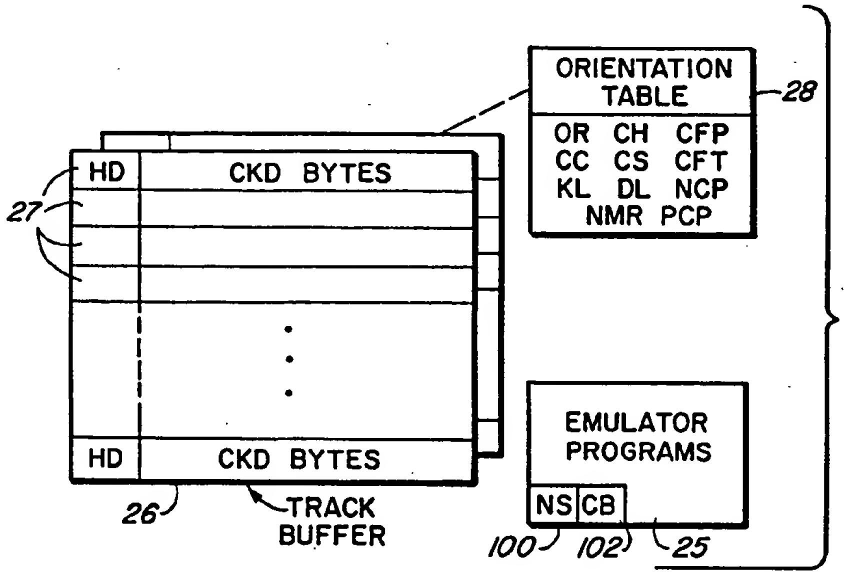


FIG. 2

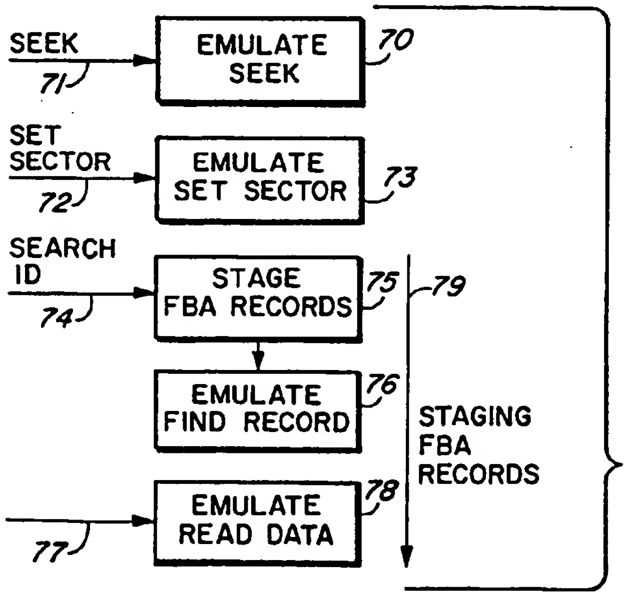


FIG. 4

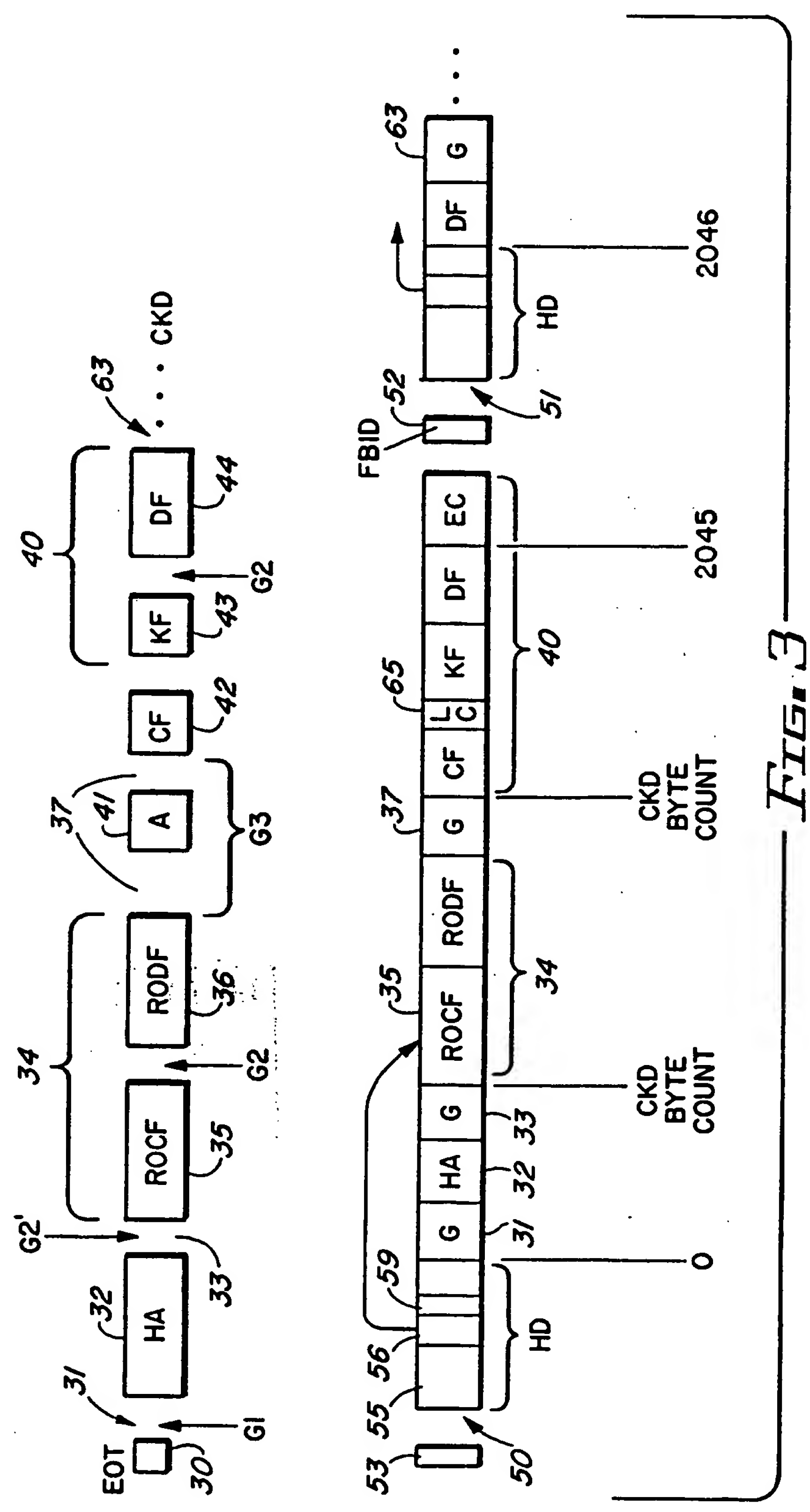

EOT
30
31
G1
HA
32
33
G2'
ROCF
35
34
G2
RODF
36
37
41
A
G3
CF
42
KF
43
40
G2
DF
44
63
CKD
53
55
56
50
59
HD
0
G
31
HA
32
G
33
CKD BYTE COUNT
ROCF
35
RODF
34
G
37
CF
L C
65
KF
40
DF
EC
CKD BYTE COUNT
2045
FBID
52
51
HD
DF
G
63
2046

Fig. 3